Modelling energy efficient server management policies in PEPA

Ali Alssaiari
School of Computing Science,
Newcastle University, UK

Ray Adderley Jm Gining
Faculty of Computer and
Mathematical Sciences,
Universiti Teknologi MARA, Malaysia

Nigel Thomas
School of Computing Science,
Newcastle University, UK
Nigel.Thomas@ncl.ac.uk

ABSTRACT

Power inefficiency has become a major concern for large scale computing providers. In this paper, we model turning servers on and off to keep a balance between capacity and energy saving. Several heuristic-based switching policies are introduced with a view to balance the cost between power saving and performance. Models are specified using a Markovian process algrebra, which allows explicit representation of system behaviour and facilitates numerical analysis using the supporting tools.

Keywords

Energy, performance modelling, PEPA.

1. INTRODUCTION

The cost of energy is one of the many challenges facing large-scale computing. According to [15], data centre owners now expect to spend more capital on energy than their IT infrastructure, which currently contributes more to the total cost of ownership (TOC). The Environmental Protection Agency (EPA) has issued a report to the U.S. Congress about the energy efficiency of servers and data centres. The report highlighted several important points related to the energy consumption of data centres. According to the report, data centre electricity demands grew 100% between 2000 and 2006. Data centres in the U.S. consumed 61 billion kWh in 2006, representing 1.5\% of total electrical consumption in the country [3]. Gartner estimate the ICT industry was responsible for 2% of global CO_2 emissions in 2007 [16] With western european data centre power consumption estimated at 56 TWh/year in 2007 and projected to double by 2020 [2], the need to improve energy efficiency of IT operations is imperative.

One of the more challenging problems in managing energy consumption in distributed systems is in handling variability of workload [9]. There are a number of measures which can be applied to manage the effect of variable supply and demand. For example, there are a variety of load balancing techniques [6] and traffic shaping measures [7] which can be utilised to manage demand so that resources do not become excessively over-utilised when demand is high. An alternative approach is to dynamically manage the supply of service capability by making more servers available during periods of high demand. [20,21] considered the problem of finding the optimal share of servers to different services under variable load in order to minimise a performance-based cost function.

This paper is based on the work of Slegers *et al* [19] and Nguyen *et al* [15]. It is focused on the notion that servers can be powered off and on according to demand in order to avoid the non-trivial energy requirements of idle servers. With perfect knowledge of arriving workload an optimal dynamic allocation of servers can be obtained which significantly reduces the overall energy demand of the system with no impact on performance, i.e. servers could be made available only when they are going to be used. Of course, we do not generally have a perfect knowledge of future workload and so an optimal dynamic solution is not practical. Instead we must investigate the trade-off between energy consumption and performance (e.g. response time) to determine the best practical method of reducing energy costs whilst not adversely affecting the quality of service. Two principle approaches to minimising energy consumption are apparent. In the first instance an optimal fixed provision of servers can be computed based on estimated workload. Depending on the variability in demand, this approach might lead to servers being idle for extended periods or to some tasks experiencing long waiting times during peak demand. The second approach is to compute a strategy to turn servers on and off based on the current (or past) state of the system. This approach minimises idle time by turning off servers, but potentially delays tasks which arrive in a burst as it takes time to turn servers back on. In addition, powering servers off and on may lead to faults which not only reduce the total available number of servers, but may also further delay an arriving task.

The remainder of this paper is organised as follows. In the next section we explain the context of this work in relation to other work on modelling server policies. In Section 3 we introduce the Markovian process algebra PEPA, which we will use to specify our models. In Section 4 we describe the system model and introduce three models of heuristic strategies for controlling the number of servers powered on and off. This is followed in Section 5 by some results of our experiments. Finally we present some conclusions and directions of further work.

2. RELATED WORK

Slegers *et al* [19] introduced a model to examine the cost of holding the job in the queue and the energy consumption cost by evaluating different heuristics of powering servers on or off. Six heuristics were introduced including Idle, static, Threshold, Semi-static, High/Low arrival period and Average Flow Heuristic. Heuristics control powering on or off servers according to job demand with different criteria. However, the model in [19] does not consider the server setup time (i.e. the time needed by a server to fully powered on or down). Moreover, the benefit of powering down servers considered only the direct impact on the power consumption by servers and ignored the cascade effect [4] (i.e. indirect energy saving in other IT component). Furthermore, it does not consider different locations of servers and assumes all servers are in one data centre location. Likewise, it assumes that all servers are homogeneous, which means that they are identical

in their components and energy consumption, which is not always the case in practice.

Mitrani [12] proposed a policy to reduce power consumption in data centre by powering down a block of servers when the service can meet the job demand without that block of servers. The model assumed the data centre consist of *N* servers where n is permanent and always on and ready to serve the job while *N-n* reserved servers can be dynamically powered on or off according to the demand. The availability of the reserved servers is controlled by two thresholds, *U* and *D*, where *U* refers to up and *D* refers to down. Reserved servers powered on as a block if the job demand increased from *U* to *U*+1 and powered off in the same fashion if job demand dropped from *D*+1 to *D*. Reserved servers consume energy while powering on or off but cannot serve the job until they fully powered on. The author assumes that a job cannot be lost when powering off reserve servers, as the job will be transferred to another server.

Mitrani [13] extended the previous model [12] by introducing multiple reserve blocks that can be turned on and off dynamically in response to different loading conditions. The aim was to investigate whether this approach reduces energy cost more than the single reserve-block approach. The result showed that the advantages of using multiple reserve blocks instead of a single reserve block are minimal. Although the small amount of saving in large-scale systems can be valuable, a single reserve-block policy is sufficient in contrast to a complicated process of finding the optimal energy saving policy.

Van Do [24] proposed a simple energy-aware policy that controls the energy consumption of physical servers and moves to a low-power consumption level (e.g. sleep state) when no virtual machines are allocated to the physical server. In addition, when virtual servers are assigned to a physical server, they start operating at a high-power consumption level. The model consists of three different dynamic mechanisms to control the allocation request of virtual servers. The first mechanism allocates the request to the physical machine that has the largest number of virtual machines, but it is not entirely loaded. In contrast, the second mechanism maps the virtual machine request to the least loaded physical server. The last scheme prioritises physical servers and numbers them from lowest to highest priority. Then, when the job request arrives, it automatically chooses the fully loaded physical server that meets the prioritising scheme to place the request and activate the virtual machine.

In our previous work [1], we presented a PEPA model that considers a variant of the high/low policy introduced in [15,19]. The maximum number of jobs is bounded at N. Arrivals into the system occur at either a high or at a low rate. Jobs leave the system according to the service process, which is determined by the number of active servers. *M* servers are static and remain permanently available to serve jobs. The remaining servers turn on and off in response to the high and low periods of arrivals. Thus, when a high period ends, these dynamic servers will become unavailable for service, but when a low period ends, they will turn back on. It is assumed that there is a delay in turning servers on and off. Therefore, when a high period begins, there will be a delay until the dynamic servers are available to serve jobs. If this delay is large and the high-arrival rate greatly exceeds the service capacity of the static servers, then there may be a significant increase in the number of jobs in the system during this time. During the turning on and turning off periods, servers will continue to consume power while not providing a service. It is further assumed that servers may fail when switching on and off. Following failures, servers undergo repair, and it is assumed that the servers will consume energy during repair as if they are working normally. The problem associated with this model is to find the optimal number of static and dynamic servers needed to minimise the energy usage for a given set of parameters (arrival rates, service rate, switching rates, failure probability and repair rate). The experiment analyses the effect of the policy on energy consumption and performance cost. Different combinations of dynamic and static servers are compared against different scenarios, including change job arrival rate, job arrival duration and the time that is needed by servers to power on fully and serve jobs. The experiment gives an interesting outcome because every scenario is unique; therefore, no specific server combination provides low-energy use and high performance in all scenarios.

Finally, there are also other existing researches [5,11,14,17,18] that have considered energy efficiency and dynamic server allocation in the relative context as previous mentioned work in this paper.

3. PEPA

A formal presentation of PEPA is given in [8] in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that occur with rates that are negative exponentially distributed. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α,r) is described by the *type* of the activity, α and the *rate* of the associated negative exponential distribution, *r*. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

$$P ::= (\alpha, r).P \mid P + Q \mid P / L \mid P \underset{L}{\bowtie} Q \mid A$$

The component $(\alpha,r).P$ performs the activity of type α at rate *r* and then behaves like *P*.

The component $P+Q$ behaves either like *P* or like *Q*, the resultant behaviour being given by the first activity to complete.

The component *P/L* behaves exactly like *P* except that the activities in the set *L* are concealed, their type is not visible and instead appears as the unknown type τ.

Concurrent components can be synchronised, such that activities in the *cooperation set L* involve the participation of both components:

$$P \underset{L}{\bowtie} Q$$

In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to that activities of that type.

Multiple unsynchronised instances of the same component can be expressed as $P[N]$. In this form it is not defined which instance of *P* will perform an action. Hence $P[2]$ is not the same as $P|P$, where we would distinguish between the derivatives $P|P'$ and $P'|P$.

$A \overset{def}{=} P$ gives the constant *A* the behaviour of the component *P*.

In the following sections only models which have a steady state solution are considered, necessary conditions for which are given in [8].

PEPA can generally be used to specify models in a concise manner using cooperating components. However, as with any formalism, there are limitations to the efficiency of specification when it comes to certain model types. In [22,23] a detailed approach to modelling queues using PEPA was presented. In [22] a queue component is specified by explicitly representing the number of items in the queue and the transitions that lead to the change in queue length, as follows.

$$Q_0 \stackrel{def}{=} (arrive, \top).Q_1$$
$$Q_1 \stackrel{def}{=} (arrive, \top).Q_2 + (serve, \top).Q_0$$
$$\ldots$$
$$Q_N \stackrel{def}{=} (serve, \top).Q_{N-1}$$

$$Arrival \stackrel{def}{=} (arrive, a).Arrival$$
$$Service \stackrel{def}{=} (serve, s).Service$$

$$Arrival \underset{\{arrive\}}{\bowtie} Q_0 \underset{\{serve\}}{\bowtie} Service$$

This means that for a queue with maximum length N, it is necessary to define N+1 expressions. Clearly if N is large, this approach is not ideal. More recently PEPA has been enhanced with a syntactic construct which makes it easier to specify multiple instances of a component as introduced above. This leads to an alternative specification of a queue as follows.

$$Q_{empty} \stackrel{def}{=} (arrive, a).Q_{full}$$
$$Q_{full} \stackrel{def}{=} (serve, s).Q_{empty}$$

$$Arrival \underset{\{arrive\}}{\bowtie} Q_{empty}[N] \underset{\{serve\}}{\bowtie} Service$$

Using this form of specification for a queue leads to a much more concise expression for large queue lengths, although some care is needed to avoid unintended calculations of the apparent rate if using passive actions. Unfortunately it is not generally possible to use this approach if it is necessary to know the number of items in a queue in order to control the behaviour of another component, for example a threshold queue length, or if there are actions which add or remove many items simultaneously (e.g. batch arrivals or queue flushing).

4. THE MODEL

Consider a system containing M homogeneous servers which can be in any one of four operational states: powered up, powered down, powering up or powering down. The powered up servers could be working or staying idle, while there were only one mode each for the other states. In the powered down mode the server is assumed not to be consuming power, although it is would be trivial to amend this to consuming power at a low rate. Jobs arrive into a bounded queue as long as the queue is not full. When the queue is full jobs are assumed to be lost. Each job in the queue is served by a single server in FIFO order. The challenge for such a system is to formulate a server management policy which reduces energy consumption but does not overly impact on response time or job loss. In this paper we will focus on policies which react to the number of jobs in the system to turn servers on or off. More specifically we define threshold values for the number of jobs in the queue where exceeding a threshold causes a server to be powered on and going below a threshold causes a server to be powered down. In general we may define different thresholds for powering up and down and possibly multiple thresholds of each type to power up or down different numbers of servers. However, finding optimal values for multiple thresholds is a non-trivial problem and can lead to behaviours which are hard to understand, even in a fairly simple system.

4.1 Static allocation policy

The first case to consider is where servers never turn off or on; instead a fixed number of servers are permanently available. This case serves as a baseline to assess the potential benefit of policies which allow servers to be dynamically managed. It also serves as a simple introduction to the kind of PEPA model we employ.

$$Q_{empty} \stackrel{def}{=} (arrive, \lambda).Q_{full}$$
$$Q_{full} \stackrel{def}{=} (serve, \mu).Q_{empty}$$

$$Arrival_{on} \stackrel{def}{=} (arrive, \lambda).Arrival_{on} + (periodOff, \beta).Arrival_{off}$$
$$Arrival_{off} \stackrel{def}{=} (periodOn, \gamma).Arrival_{on}$$

$$Server \stackrel{def}{=} (serve, \mu).Server$$

$$Arrival_{on} \underset{\{arrive\}}{\bowtie} Q_{empty}[N] \underset{\{serve\}}{\bowtie} Server[M]$$

The queue component (starting with all places empty) has N possible places. Arrivals alternate between being *on* and *off*, switching according to the actions *periodOn* and *periodOff*. When the arrivals are on, they occur at rate λ. The M servers are fixed and offer service at all times; the rate at which the *serve* action occurs being min(j,M), where j is the number of jobs in the queue ($0 \leq j \leq N$).

4.2 Semi-static allocation policy

A modification to the static policy is to have some servers which are always available and some which power off when the arrivals turn off.

$$Server_{static} \stackrel{def}{=} (serve, \mu).Server_{static}$$
$$Server_{on} \stackrel{def}{=} (serve, \mu).Server_{on} + (periodOff, \beta).ServerPowerOff$$
$$ServerPowerOff \stackrel{def}{=} (powerOff, \xi).Server_{off}$$
$$Server_{off} \stackrel{def}{=} (periodOn, \gamma).ServerPowerOn$$
$$ServerPowerOn \stackrel{def}{=} (powerOn, \eta).Server_{on}$$
$$Servers \stackrel{def}{=} (Server_{on} \underset{\{periodOn,\ periodOff\}}{\bowtie} Server_{on})$$
$$((Arrival_{on} \underset{\{periodOn,\ periodOff\}}{\bowtie} Servers) \underset{\{\}}{\bowtie} Server_{static}[M-m]) \underset{\{arrive,\ serve\}}{\bowtie} Q_{empty}[N]$$

The queue and arrival components are specified as in the previous case. The servers are divided between M-m static servers, which are specified as in the static allocation policy, and m dynamic servers which can power on and off (in the above case m=2). It is assumed that changing between power modes takes time, hence

there are intermediate states *ServerPowerOn* and *ServerPowerOff*. When a server is on it may serve jobs, but at the end of the on period of arrivals, all dynamic servers start to power down. Similarly when the arrivals off period ends, all the dynamic servers begin to power up. Note that the dynamic servers synchronise over the *periodOn* and *periodOff* actions to ensure that all servers initiate the change of mode at the same time. However they each complete the mode transition independently. This has two implications. Firstly it means that the arrival mode changes can only happen when all dynamic servers are on or off. Given that the periods of the arrival modes are generally much longer than the switching times of servers, this is a small consideration. The second implication is that there will be a small time after the arrivals have turned on before all the dynamic servers are available, which may lead to a temporary overload.

4.3 Threshold policy

In the previous policy the dynamic servers responded to changes in the behaviour of arrivals. This might be feasible when there are well understood arrival modes which are readily detected. A simpler mechanism is to employ a threshold on the number of jobs in the queue to determine when to turn servers on or off. In order to specify such a policy in PEPA, we need to keep track of the number of jobs in the queue, which means we cannot use the same queue specification as above.

$$Q_0 \overset{def}{=} (arrive, \lambda).Q_1 + (turnOff, \theta).Q_0$$

...

$$Q_j \overset{def}{=} (arrive, \lambda).Q_{j+1} + (serve,\ j\mu).Q_{j-1} + (turnoff, \theta).Q_j$$

$$Q_{j+1} \overset{def}{=} (arriveT, \lambda).Q_{j+2} + (serve, (j+1)\mu).Q_j$$

...

$$Q_N \overset{def}{=} (serve, N\mu).Q_{N-1}$$

This queue component has a threshold value at *j*; if there *j* jobs or fewer in the queue then any dynamic servers will power off. Note that to preserve the correct service rate over multiple servers the rate specified at the queue is $i\mu$, so that the actual rate will be the minimum of this value and μ times the number of available servers. When the number of jobs in the queue exceeds *j*, the name of the arrival action changes to *arriveT*. This acts as a trigger action for the dynamic servers to turn on. First we need to modify the arrival process as follows.

$$Arrival_{on} \overset{def}{=} (arrive, \lambda).Arrival_{on} + (periodOff, \beta).Arrival_{off} + (arriveT, \lambda).Arrival_{on}$$

$$Arrival_{off} \overset{def}{=} (periodOn, \gamma).Arrival_{on}$$

As *arrive* and *arriveT* cannot occur concurrently in the queue, this does not change the mathematical properties of the arrival process, just the name of the current arrival action. Any static servers are specified as previously. Dynamic servers are specified in a very similar way as previously, with the exception that it is the *turnoff* action which initiates powering off and the *arriveT* action which initiates powering on. Unlike the previous case, we do not synchronise the dynamic servers, so each *arriveT* action will turn on one server only. Similarly servers turn off one by one. This independence makes some sense as the servers are no longer responding to long term changes in arrival behaviour, but rather the arrivals of single jobs. Intuitively it would not make much sense to turn on *m* more servers just because there is one more jobs to serve.

$$Server_{on} \overset{def}{=} (serve, \mu).Server_{on} + (turnOff, \theta).ServerPowerOff + (arriveT, \lambda).Server_{on}$$

$$ServerPowerOff \overset{def}{=} (powerOff, \xi).Server_{off} + (arriveT, \lambda).Server_{on}$$

$$Server_{off} \overset{def}{=} (arriveT, \lambda).ServerPowerOn$$

$$ServerPowerOn \overset{def}{=} (powerOn, \eta).Server_{on} + (arriveT, \lambda).ServerPowerOn$$

$$(Arrival_{on} \underset{K}{\bowtie} Queue_0) \underset{L}{\bowtie} (Server_{static}[M-m] \underset{\{\}}{\bowtie} Server_{on}[m])$$

4.4 Cost function

We now need some metric or metrics by which to compare these different policies. Intuitively the static policy will have the largest power consumption but the best performance, but the other two policies are less clear. We define a simple cost function which is the sum of an energy cost plus a performance cost for each policy. The energy cost is defined as a constant, C_1, times the number of servers which are not off. This assumes that all servers consume the same power whether they are active, idle or powering on or off. The performance cost is defined as a constant, C_2, times the average queue length. This function is a reasonable discriminator of system performance as long as the probability of the queue being full is small.

5. Numerical results

First we consider the overall cost of the policies under different arrival rates.

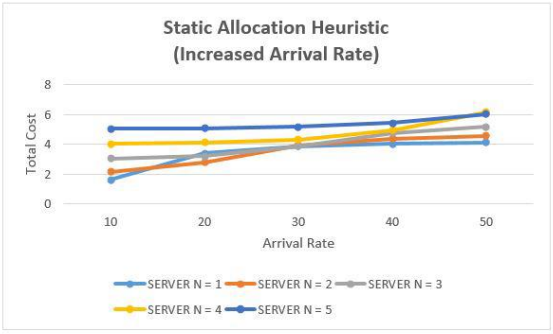

Figure 1. Cost of static policy varied with arrival rate, λ.

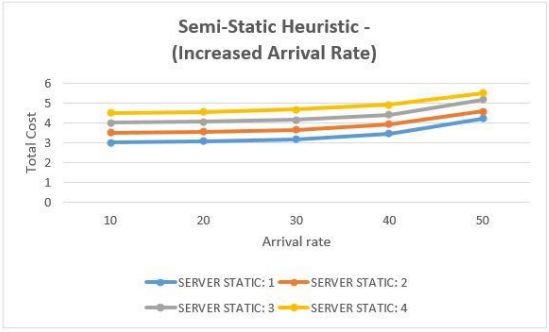

Figure 2. Cost of semi-static policy varied with arrival rate, λ.

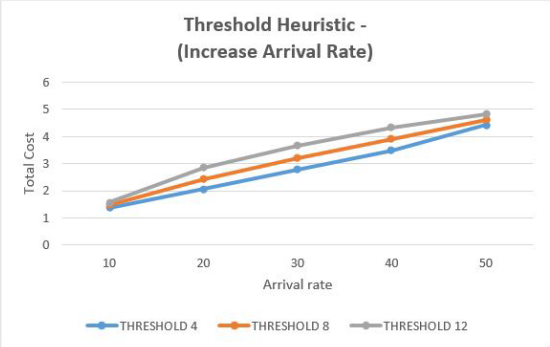

Figure 3. Cost of threshold policy varied with arrival rate, λ.

In these experiments there were a maximum of 5 servers, each with a service rate of 10. The maximum queue length was 20 and the number of static servers in the threshold policy, *m*, was always 1. The rates of powering up and down were also 10 and the periods of arrivals being on or off were equal, with rate 0.5. The costs used were C_1=1 and C_2=0.25. In Figure 1 we see that the energy cost dominates and hence it is preferable to have fewer available servers. However, when the arrival rate is high the system will overload during arrival on periods when there are too few servers. In this case the performance cost does not distinguish between the different cases and so the overall cost is determined by the least number of servers only. If the maximum queue length was greater, then this might lead to a greater distinction as load increases. In Figure 2 we observe a similar picture, where the least number of static servers is considered to be the best configuration. In this policy all the other servers are available when the arrivals are on, so the overload situation is not as severe, although the dominance of the energy cost and the relatively short queue length means that the overall cost is no better than the static case. In Figure 3 we compare three different threshold values. The smallest threshold (*j*=4) gives a slightly better overall cost, although when the arrival rate is highest most of the servers are on most of the time when the arrivals are on, so there is little difference with the semi-static case. The best configuration of each policy is shown in Figure 4, compared against the naïve policy of all servers being on.

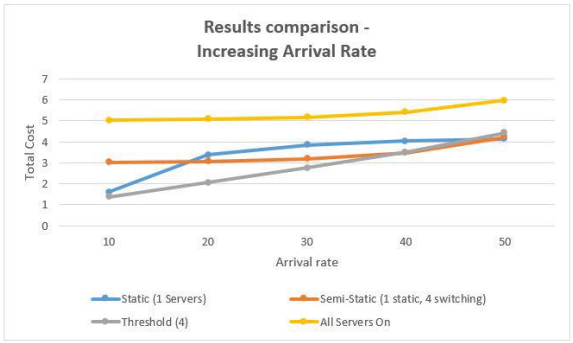

Figure 4. Cost of all policies varied with arrival rate, λ.

It is clear from the above results that the performance costs did not have much impact on the choice of optimal configuration. In Figure 5 we present the same policies and configurations as Figure 4, but consider different values for the weight of the performance cost. The arrival rate here is 40.

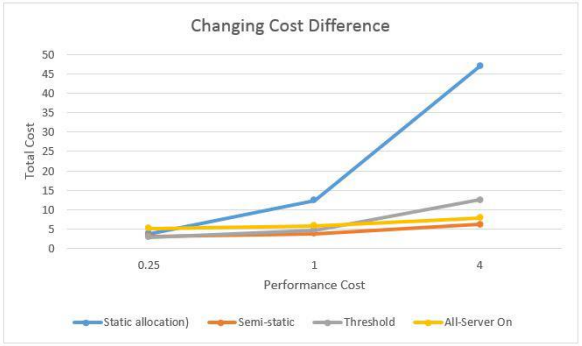

Figure 5. Cost of all policies against performance weight C_2.

Clearly increasing the performance cost has a dramatic effect on the static policy with only one server. This is because in this configuration the queue is full for most of the time when the arrivals are on and so is penalized when the performance weight is larger. As we have seen above, there is little to choose between the threshold and semi-static policies when the performance weight is small, but it is slightly surprising that the threshold policy performs worse as C_2 becomes larger. The explanation here is that when the queue length drops below the threshold, most of the servers switch off, which means that the queue length immediately grows again, causing them to switch back on. As this switching wastes time (and energy) that could be used to serve jobs, the result is a poorer performance than the semi-static policy where all the dynamic servers switch only at the start and end of the arrival on period. A better performance of the threshold policy might be achieved by choosing a larger value of *m*.

Finally, in Figure 6, we consider the effect of changing the length of the arrival on and off periods. The experimental set up is the same as Figure 5, except that the rate of the on/off periods is increased on a log scale from 0.1 to 100 (thus decreasing the on/off period duration). The threshold *j*=8 and there are two dynamic servers and three static servers in the semi-static case. As with Figures 1-4, the performance weight C_2=0.25.

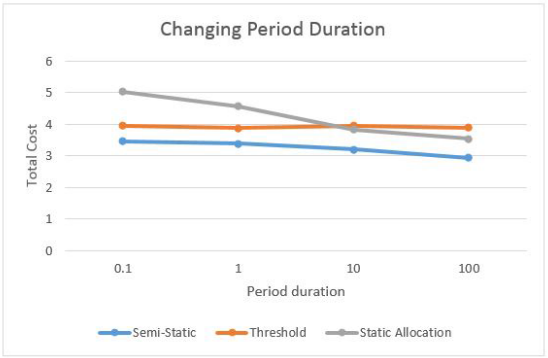

Figure 5. Cost of all policies against arrival on period rate.

When the period is long (rate is 0.1) the static policy performs predictably poorly, as for most of the off period, there are numerous idle servers. The threshold policy is shown to give the most stable cost across the range of rates, which is not surprising given that it is the most flexible of the policies investigated. The best performance is given by the semi-static policy, which manages to balance the stability of the static policy with some of

the flexibility of the threshold policy. However, it is worth noting that if the arrival rate was higher, and the performance weight was larger, then the semi-static policy might struggle by powering off servers at the end of the arrival on period when the queue is still large.

6. Conclusions and Further work

In this paper we have shown how PEPA can be used to model policies for controlling the power mode of parallel servers. These models are conceptually simple, but determining which policy gives the least cost and why, is not always intuitive. The experiments described here are quite limited by the necessity of brevity. It would be clearly beneficial to consider larger maximum queue sizes and greater numbers of servers. The policies themselves are also quite limited. The threshold policy in particular is only defined with one threshold to turn on and off servers, which can result in an oscillating behaviour as the queue size grows and shrinks around this threshold value. Having different thresholds for turning on or off, or even multiple thresholds for different classes of dynamic servers, could add more stability to the system and improve performance.

It is worth noting that Slegers *et al* [21] proposed six policies and we have only considered three in this paper. Furthermore Nguyen *et al* [15] considered the case where servers may breakdown when powering up or down. Modelling these additional cases, as well as others from the literature, in PEPA would enable more comparisons to be made. While these models are interesting and provide some insight with regard to general principles, they are crude abstractions of real system behaviour. Therefore investigating models which are more realistic and using real data where possible, is clearly desirable.

7. REFERENCES

[1] Alssaiari, A., and N. Thomas, Performance modelling of dynamic server allocation for energy efficiency using PEPA, 32nd UK Performance Engineering Workshop, University of Bradford, 2016.

[2] Bertoldi, P. and B. Anatasiu., Electricity Consumption and Efficiency Trends in European Union Status Report, 2009.

[3] Brown, R., Report to congress on server and data center energy efficiency: Public law 109-431, Lawrence Berkeley National Laboratory, 2008.

[4] Emerson Network Power, Energy logic: Reducing data center energy consumption by creating savings that cascade across systems, a white paper from the experts in business-critical continuity, RCI HI RCI, 20:40-60, 2013.

[5] Gandhi, A., M. Harchol-Balter and I. Adan, Server farms with setup costs, Performance Evaluation, **67**(11), 1123-1138, 2010.

[6] Gilly, K., C. Juiz, N. Thomas and R. Puigjaner, Scalable QoS content-aware load balancing algorithm for a Web Switch based on classical policies, 22nd IEEE International Conference on Advanced Information Networking and Applications', IEEE, 2008.

[7] Gilly, K., C. Juiz, N. Thomas and R. Puigjaner, Adaptive Admission Control Algorithm in a QoS-aware Web System, Journal of Information Sciences, **199**, 58-77, 2012.

[8] Hillston, J., A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.

[9] Jarvis, A., N. Thomas and A. van Moorsel, Open issues in grid performability, International Journal of Simulation, **5**(5): 3-12, 2004.

[10] Kleinrock, L., Queueing systems, Volume I: Theory, 1975.

[11] Maccio, V.J. and D.G. Down, On optimal policies for energy-aware servers, 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 31-39, 2013.

[12] Mitrani, I., Service center trade-offs between customer impatience and power consumption, Performance Evaluation, **68**(11): 1222-1231, 2011.

[13] Mitrani, I., Trading power consumption against performance by reserving blocks of servers, 9th European Workshop on Performance Engineering, LNCS 7587, Springer, 2012.

[14] Mitrani, I., Managing performance and power consumption in a server farm, Annals of Operations Research, **202**(1): 121-134, 2013.

[15] Nguyen, T.H., M. Forshaw, and N. Thomas, Operating policies for energy efficient dynamic server allocation. Electronic Notes in Theoretical Computer Science, **318**, 159-177, 2015.

[16] Pettey, C., Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions, 2007. http://www.gartner.com/newsroom/id/503867

[17] Phung-Duc, T., Controllable setup queue for energy-aware server, 31st UK Performance Engineering Workshop, University of Leeds, 2015.

[18] Phung-Duc, T., Impatient customers in power-saving data centers, in: International Conference on Analytical and Stochastic Modeling Techniques and Applications, LNCS 8499, Springer, 2014.

[19] Slegers, J., N. Thomas and I. Mitrani, Static and dynamic server allocation in systems with on/off sources, Annals of Operations Research, **170**(1), 251-263, 2009.

[20] Slegers, J., N. Thomas and I. Mitrani, Evaluating the optimal server allocation policy for clusters with on/off sources, Performance Evaluation, **66**(8), 453-467, 2009.

[21] Slegers, J., N. Thomas and I. Mitrani, Dynamic server allocation for power and performance, in: Performance Evaluation: Metrics, Models and Benchmarks, LNCS 5119, Springer, 2008.

[22] Thomas, N. and J. Hillston, Using Markovian process algebra to specify interactions in queueing systems, 14th UK Performance Engineering Workshop, University of Edinburgh, 1998.

[23] Thomas, N. and S. Gilmore, Applying quasi-separability to Markovian process algebra, 6th International Workshop on Process Algebra and Performance Modelling, Nice, 1998.

[24] Van Do, T., Comparison of allocation schemes for virtual machines in energy-aware server farms, The Computer Journal, **54**(11): 1790-1797, 2011.